39장. DOM⭐

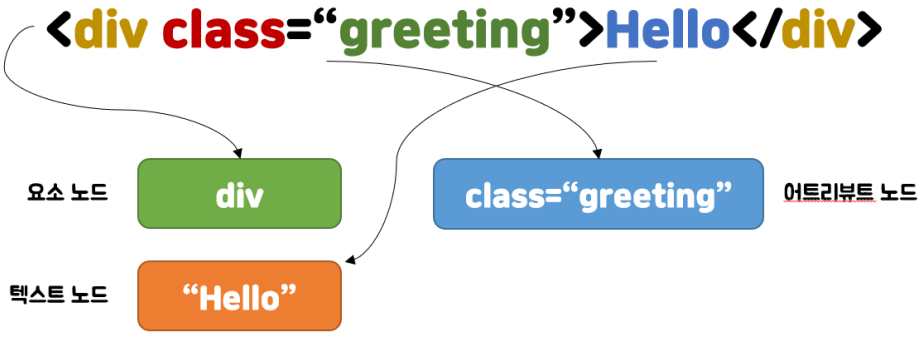
1. **DOM(Document Object Model)은 HTML 문서의 계층적 구조와 정보를 표현하며 이를 제어할 수 있는 API, 즉 프로퍼티와 메서드를 제공하는 트리 자료구조.**

39.1 노드

39.1.1 HTML 요소와 노드 객체

1. HTML 요소(HTML element) : HTML 문서를 구성하는 개별적인 요소

- HTML 요소는 렌더링 엔진에 의해 파싱되어 DOM을 구성하는 요소 노드 객체로 변환.



- HTML 요소 간의 부자 관계를 반영하여 HTML 문서의 구성 요소인 HTML 요소를 객체화한 모든 객체들을 트리 자료 구조로 구성.

트리 자료구조

- 트리 자료구조는 노드들의 계층 구조로 이뤄진다.
- **노드 객체들로 구성된 트리 자료구조를 DOM이라 한다.**

  노드 객체의 트리로 구조화되어 있기 때문에 DOM을 **DOM트리**라고 부르기도 한다.

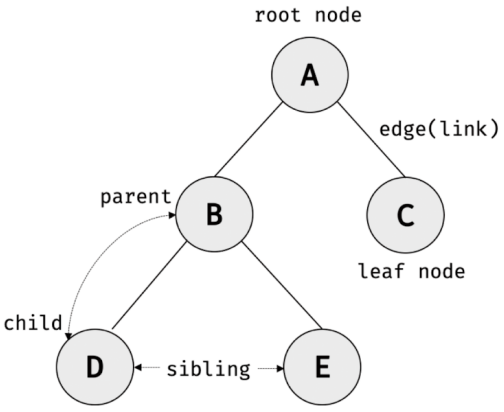


그림 39–3 트리 자료구조

39.1.2 노드 객체의 타입

- DOM은 노드 객체의 계층적인 구조로 구성됨.

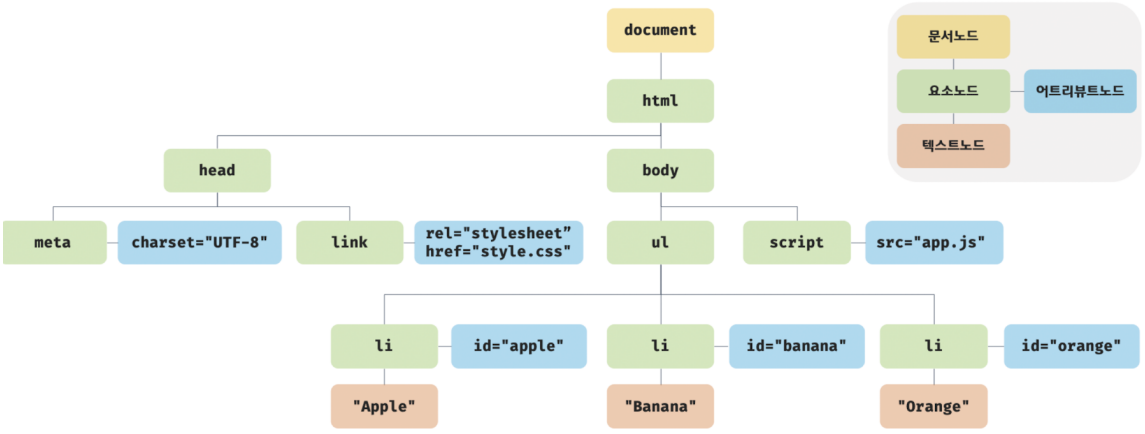


그림 39–4 DOM

1. 문서 노드

- 문서 노드는 DOM 트리의 최상위에 존재하는 루트 노드로서 document 객체를 가리킴.
- 문서 노드, 즉 document 객체는 DOM 트리의 루트 노드이므로 DOM 트리의 노드들에 접근하기 위한 **진입점 역할**을 담당.
- 즉, 요소, 어트리뷰트, 텍스트 노드에 접근하려면 문서 노드를 통해야 한다.

2. 요소 노드

- 요소 노드는 HTML 요소를 가리키는 객체.
- 따라서 요소 소드는 **문서의 구조를 표현**한다고 할 수 있다.

3. 어트리뷰트 노드

- 어트리뷰트 노드는 HTML 요소의 어트리뷰트를 가리키는 객체.
- 요소 노드는 부모 노드와 연결되어 있지만 어트리뷰트 노드는 부모 노드와 연결되어 있지 않고 요소 노드에만 연결됨.
- 따라서 어트리뷰트 노드에 접근하여 어트리뷰트를 참조하거나 변경하려면 먼저 **요소 노드에 접근**.

4. 텍스트 노드

- 텍스트 노드는 HTML 요소의 텍스트를 가리키는 객체다.
- 텍스트 노드는 요소의 자식노드이며, DOM 트리의 최종단이다.
- 따라서 텍스트 노드에 접근하려면 **부모 노드인 요소 노드에 접근**해야 함.

39.1.3 노드 객체의 상속 구조

- DOM을 구성하는 노드 객체는 자신의 구조와 정보를 제어할 수 있는 **DOM API**를 사용할 수 있다.

  → 이를 통해 자신의 부모, 형제, 자식을 탐색 가능
- 노드 객체의 상속 구조

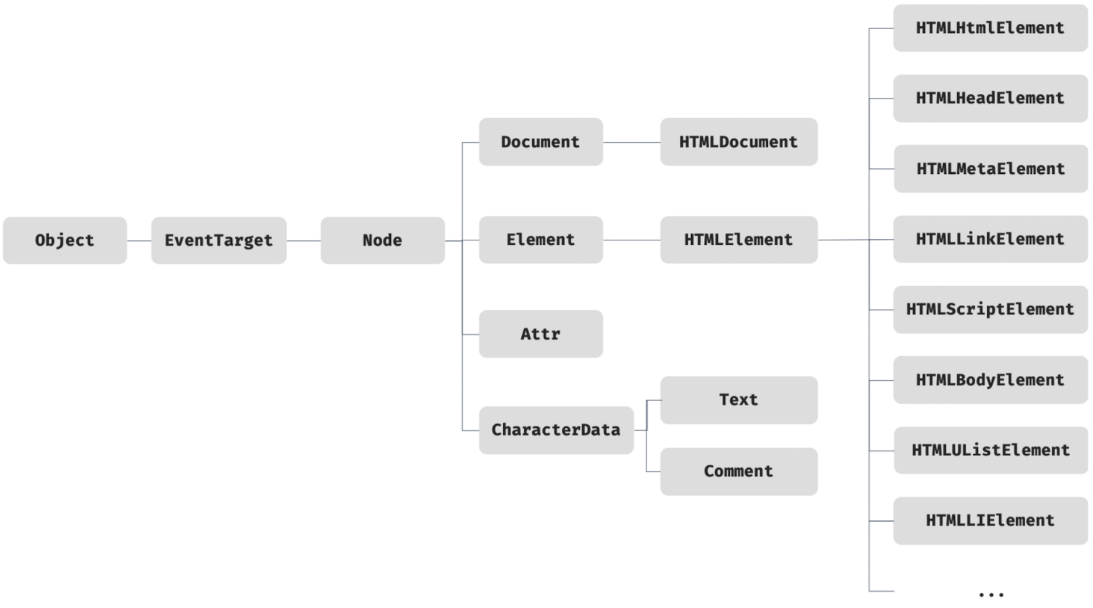


그림 39–5 노드 객체의 상속 구조

- 모든 노드 객체는 Object, EventTarget, Node 인터페이스를 상속받는다.

결론.

DOM은 HTML 문서의 계층적 구조와 정보를 표현하는 것은 물론 노드 객체의 종류, 즉 노드 타입에 따라 필요한 기능을 프로퍼티와 메서드의 집합인 DOM API로 제공. 이 DOM API를 통해 HTML의 구조나 내용 또는 스타일 등을 동적으로 조작할 수 있다.

💡 중요한 것은 DOM API, 즉 DOM이 제공하는 프로퍼티, 메서드를 사용하여 노드에 접근하고 HTML의 구조나 내용 or 스타일 등을 동적으로 변경하는 방법을 익히는 것.

FE 개발자에게 HTML은 단순히 태그와 어트리뷰트를 선언적으로 배치하여 뷰를 구성하는 것 이상의 의미를 갖는다. 즉,
HTML을 DOM과 연관 지어 바라볼 것.

39.2 요소 노드 취득

39.2.1 id를 이용한 요소 노드 취득

- Document.prototype.getElementById 메서드는 인수로 전달한 id 어트리뷰트 값(id 값)을 갖는 하나의 요소 노드를 탐색하여 반환한다.
- 반드시 문서 노드인 document를 통해 호출해야한다.

```
const $elem = document.getElementById('id');
```

1. id 값은 HTML 문서 내에서 유일한 값이어야한다. 만약 중복된 id가 여러 개 존재하더라도 어떠한 에러도 발생 X

   이러한 경우 getElementById 메서드는 인수로 전달된 id 값을 갖는 첫 번째 요소 노드만 반환
2. id 값을 갖는 HTML 요소가 존재하지 않는 경우 null 반환.
3. HTML 요소에 id 어트리뷰트를 부여하면 id 값과 동일한 이름의 전역 변수가 암묵적으로 선언되고 해당 노드 객체가 할당되는 부수 효과가 있다.
4. id 값과 동일한 전역 변수가 이미 선언되어 있으면 이 전역 변수에 노드 객체가 재할당 되지 X

39.2.2 태그 이름을 이용한 요소 노드 취득

- Document.prototype/Element.prototype.getElementsByTagName 메서드

  : 인수로 전달한 태그 이름을 갖는 모든 요소 노드들을 탐색하여 반환
- 여러 개의 요소 노드 객체를 갖는 DOM 컬렉션 객체인 HTMLCollection 객체를 반환

```
<!DOCTYPE html>
<html>
  <body>
    <ul>
      <li id="apple">Apple</li>
      <li id="banana">Banana</li>
      <li id="orange">Orange</li>
    </ul>
    <script>
        // 태그 이름이 li인 요소 노드를 모두 탐색하여 반환
      // HTMLCollection 객체는 유사 배열 객체이면서 이터러블이다.
```

```
      const $elem = document.getElementsByTagName('li');

            // 취득한 모든 요소 노드의 style.color 프로퍼티 값을 변경
      // HTMLCollection 객체를 배열로 변환하여 순회하면서 color 프로퍼티 값을 변경한
다.
      [...$elem].forEach(elem => { elem.style.color = 'red'; });
    </script>
  </body>
</html>
```

- HTML 문서의 모든 요소 노드를 취득하려면 인수로 `'*'`를 전달
- **Document.prototype.getElementsByTagName** 메서드는 DOM의 루트 노드인 문서 노드, 즉 document를 통해 호출하며 DOM 전체에서 요소 노드를 탐색하여 반환한다.
- **Element.prototype.getElementsByTagName** 메서드는 특정 요소 노드를 통해 호출하며, 특정 요소 노드의 자손 노드 중에서 요소 노드를 탐색하여 반환한다.

```
<!DOCTYPE html>
<html>
  <body>
    <ul id="fruits">
      <li>Apple</li>
      <li>Banana</li>
      <li>Orange</li>
    </ul>
    <ul>
      <li>HTML</li>
    </ul>
    <script>
      // DOM 전체에서 태그 이름이 li인 요소 노드를 모두 탐색하여 반환한다.
      const $lisFromDocument = document.getElementsByTagName('li');
      console.log($lisFromDocument); // HTMLCollection(4) [li, li, li, li]

      // ul#fruits 요소의 자손 노드 중에서 태그 이름이 li인 요소 노드를 모두 탐색하여
반환
      const $fruits = document.getElementById('fruits');
      const $lisFromFruits = $fruits.getElementsByTagName('li');
      console.log($lisFromDocument); // HTMLCollection(3) [li, li, li]
    </script>
  </body>
</html>
```

- 인수로 전달된 태그 이름을 갖는 요소가 존재하지 않는 경우 빈 객체 반환

39.2.3 class를 이용한 요소 노드 취득

- Document.prototype/Element.prototype.getElementsByClassName 메서드는 인수로 전달한 class 어트리뷰트 값을 갖는 모든 요소 노드들을 탐색하여 반환.
- 여러 개의 요소 노드 객체를 갖는 DOM 컬렉션 객체인 HTMLCollection 객체를 반환한다.

```
const $elems = document.getElementsByClassName('fruit');
```

- **Document.prototype.getElementsByClassName** 메서드는 DOM의 루트 노드인 문서 노드, 즉 document를 통해 호출하며 DOM 전체에서 요소 노드를 탐색하여 반환한다.
- **Element.prototype.getElementsByClassName** 메서드는 특정 요소 노드를 통해 호출하며, 특정 요소 노드의 자손 노드 중에서 요소 노드를 탐색하여 반환한다.
- 인수로 전달된 태그 이름을 갖는 요소가 존재하지 않는 경우 빈 객체 반환

39.2.4 CSS 선택자를 이용한 요소 노드 취득

- CSS 선택자(selector)

  : 스타일을 적용하고자 하는 HTML 요소를 특정할 때 사용하는 문법

1. Document.prototype/Element.prototype.querySelector 메서드는 인수로 전달한 CSS 선택자를 만족시키는 하나의 요소 노드를 탐색하여 반환한다.
   - 만족시키는 요소 노드가 여러 개인 경우 첫 번째 요소 노드 반환
   - 만족시키는 요소 노드가 존재하지 않는 경우 null을 반환
   - 문법에 맞지 않는 경우 DOMException 에러 발생

```
// class 어트리뷰트 값이 'banana'인 첫 번째 요소 노드 탐색하여 반환
const $elem = document.querySelector('.banana');
```

2. Document.prototype/Element.prototype.querySelectorAll 메서드는 임수로 전달한 CSS 선택자를 만족시키는 모든 요소 노드를 탐색하여 반환한다.

- 여러 개의 요소 노드 객체를 갖는 DOM 컬렉션 객체인 NodeList 객체를 반환
  - 만족 시키는 요소가 존재하지 않는 경우 빈 NodeList 객체를 반환
  - 문법에 맞지 않는 경우 DOMException 에러 발생

```
// ul 요소의 자식 요소인 li 요소를 모두 탐색하여 반환
const $elems = document.querySelectorAll('ul > li');
```

결론.

id 어트리뷰트가 있는 요소 노드를 취득하는 경우에만 getElementById 메서드를 사용하고

그 외의 경우에는 querySelector, querySelectorAll 메서드를 사용하는 것을 권장한다.

39.2.5 특정 요소 노드를 취득할 수 있는지 확인

- Element.prototype.matches 메서드

  : 인수로 전달한 CSS 선택자를 통해 특정 요소 노드를 취득할 수 있는지 확인한다.

```
<!DOCTYPE html>
<html>
  <body>
    <ul id="fruits">
      <li class="apple">Apple</li>
      <li class="banana">Banana</li>
      <li class="orange">Orange</li>
    </ul>
    <script>
      const $apple = document.querySelector('.apple');

      // $apple 노드는 #fruits > li.apple 로 취득할 수 O
      console.log($apple.matches('#fruits > li.apple')); // true

      // $apple 노드는 #fruits > li.banana 로 취득할 수 X
      console.log($apple.matches('#fruits > li.banana')); // false
    </script>
  </body>
</html>
```

- 이벤트 위임을 사용할 때 유용

39.2.6 HTMLCollection과 NodeList

- HTMLCollection과 NodeList의 중요한 특징은 노드 객체의 상태 변화를 실시간으로 반영하는 **살아있는 (live) 객체**라는 것
- **HTMLCollection**은 언제나 live 객체로 동작
- **NodeList**는 대부분의 경우, 노드 객체의 상태 변화를 실시간으로 반영하지 않고 과거의 정적 상태를 유지하는 non-live 객체로 동작하지만, 경우에 따라 live 객체로 동작할 때가 있다.

HTMLCollection

- getElementsByTagName, getElementByClassName 메서드가 반환하는 HTMLCollection 객체는 노드 객체의 상태 변화를 실시간으로 반영하는 살아 있는 DOM 컬렉션 객체다.

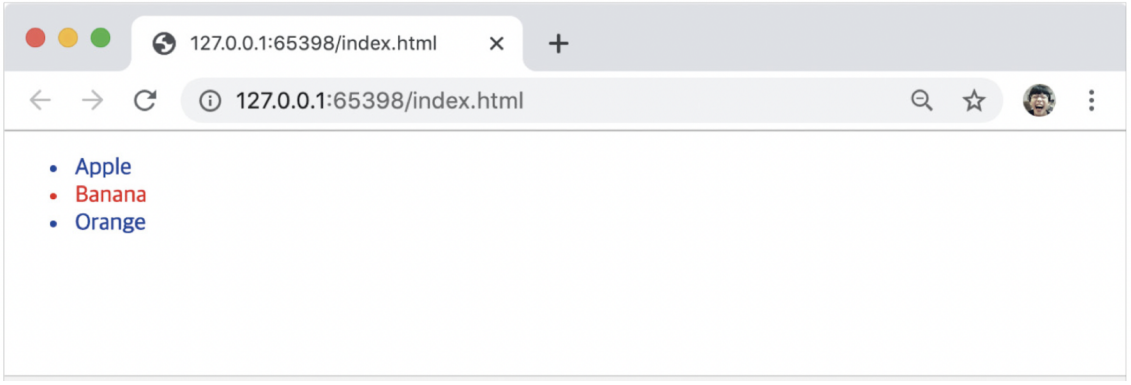
```
<!DOCTYPE html>
  <head>
    <style>
      .red { color: red; }
      .blue { color: blue; }
    </style>
  </head>
```

```html
<html>
  <body>
    <ul id="fruits">
      <li class="red">Apple</li>
      <li class="red">Banana</li>
      <li class="red">Orange</li>
    </ul>
    <script>
      // class 값이 red 인 요소 노드를 모두 탐색하여 HTMLCollection 객체에 담아 반
환.
      const $elems = document.getElementsByClassName('red');
      // 이 시점에 HTMLCollection 객체에는 3개의 요소 노드가 담겨 있다.
      console.log($elems) // HTMLCollection(3) [li.red, li.red, li.red]

      // HTMLCollection 객체의 모든 요소의 class 값을 blue로 변경한다.
      for (let i = 0; i < $elems.length; i++) {
        $elems[i].className = 'blue';
      }

      // HTMLCollection 객체의 요소가 3개에서 1개로 변경되었다.
      console.log($elems); // HTMLCollection(1) [li.red]
    </script>
  </body>
</html>
```

- 위 예제를 봤을 때 모든 값이 blue로 변경된다고 예상하지만 **그렇지 않다.**



1. 첫 번째 반복(i === 0)
2. 두 번째 반복(i === 1)
3. 세 번째 반복(i === 2)

→ HTMLCollection 객체는 실시간으로 노드 객체의 상태 변경을 반영하여 요소를 제거할 수 있기 때문에 HTMLCollection 객체를 for 문으로 순회하면서 노드 객체의 상태를 변경해야 할 때 **주의**해야 한다.

- 이 문제는 for 문을 역방향으로 순회하는 방법으로 회피할 수 있다.

```
// for 문을 역방향으로 순회
for (let i = $elems.length - 1; i >= 0; i--) {
  $elems[i].className = 'blue';
}
```

- 또는 while문을 사용하여 HTMLCollection 객체에 노드 객체가 남아 있지 않을 때까지 무한 반복하는 방법으로 회피할 수 있다.

```
// while 문으로 HTMLCollection에 요소가 남아 있지 않을 때까지 무한 반복
let i = 0;
while ($elems.length > i) {
  $elems[i].className = 'blue';
}
```

- 더 간단한 해결책은 부작용을 발생시키는 원인인 HTMLCollection 객체를 사용하지 않는 것이다.
- **배열의 고차 함수(forEach, map, filter, reduce 등)를 사용할 수 있다.**

```
// 유사 배열 객체이면서 이터러블인 HTMLCollection을 배열로 변환하여 순회
[...$elems].forEach(elem => elem.className = 'blue');
```

NodeList

- HTMLCollection 객체의 부작용을 해결하기 위해 querySelectorAll 메서드를 사용하는 방법도 있다.
- 이때 NodeList 객체는 실시간으로 노드 객체의 상태 변경을 반영하지 않는(non-live) 객체

```
// DOM 컬렉션 객체인 NodeList를 반환
const $elems = document.querySelectorAll('.red');

// NodeList 객체는 forEach 메서드를 상속받아 사용할 수 있다.
$elems.forEach(elem => (elem.className = 'blue'));
```

- **하지만 childNodes 프로퍼티가 반환하는 NodeList 객체는 live 객체로 동작하므로 주의가 필요하다.**
- 이처럼 예상과 다르게 동작할 때가 있어 다루기 까다롭고 실수하기 쉽다.
- 따라서 **노드 객체의 상태 변경과 상관 없이 안전하게 DOM 컬렉션을 사용하려면, HTMLCollection이나 NodeList 객체를 배열로 변환하여 사용하는 것을 권장한다.**

39.3 노드 탐색

- 요소 노드를 취득한 다음, 취득한 요소 노드를 기점으로 DOM 트리를 옮겨 다니며 부모, 형제, 자식 노드 등을 탐색해야 할 때가 있다.

```
<ul id="fruits">              // 3개의 자식 요소
  <li class="apple">Apple</li>
```

```
  <li class="banana">Banana</li>
  <li class="orange">Orange</li>
</ul>
```

- DOM 트리 상의 노드를 탐색할 수 있도록 Node, Element 인터페이스는 트리 탐색 프로퍼티를 제공한다.

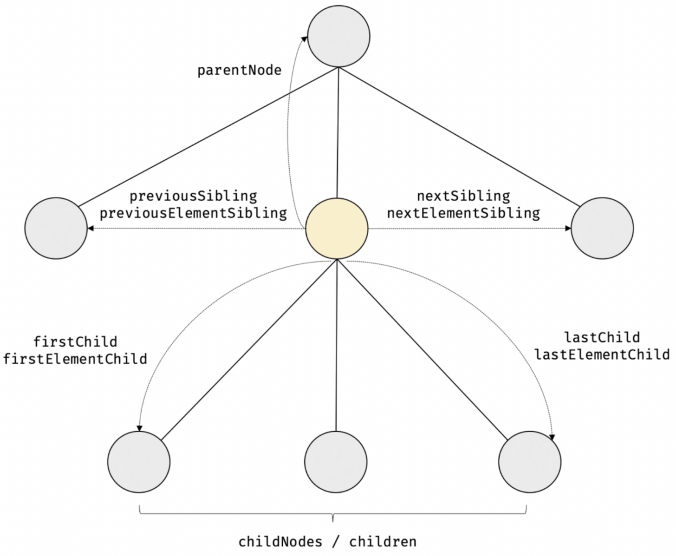


그림 39–10 트리 노드 탐색 프로퍼티

- parentNode, previousSibling, firstChild, childNodes 프로퍼티는 **Node.prototype**이 제공
- previousElementSibling, nextElementSibling, children 프로퍼티는 **Element.prototype**이 제공
- 노드 탐색 프로퍼티는 모두 접근자 프로퍼티다.

  단 setter없이 getter만 존재하는 **읽기 전용 접근자 프로퍼티**.

39.3.1 공백 텍스트 노드

- HTML 요소 사이의 스페이스, 탭, 줄바꿈 등의 공백 문자는 텍스트 노드를 생성하며 이를 **공백 텍스트 노드**라 한다.

39.3.2 자식 노드 탐색

- 자식 노드를 탐색하기 위해서는 다음과 같은 노드 탐색 프로퍼티를 사용한다.

프로퍼티	설명
Node.prototype.childNodes	자식 노드를 모두 탐색하여 DOM 컬렉션 객체인 NodeList에 담아 반환한다. **childNodes 프로퍼티가 반환한 NodeList에는 요소 노드뿐만 아니라 텍스트 노드도 포함되어 있을 수 있다.**
Element.prototype.children	자식 노드 중에서 요소 노드만 모두 탐색하여 DOM 컬렉션 객체인 HTMLCollection에 담아 반환한다. **children 프로퍼티가 반환한 HTMLCollection에는 텍스트 노드가 포함되지 않는다.**

프로퍼티	설명
Node.prototype.firstChild	첫 번째 자식 노드를 반환한다. firstChild 프로퍼티가 반환한 노드는 텍스트 노드이거나 요소 노드다.
Node.prototype.lastChild	마지막 자식 노드를 반환한다. lastChild 프로퍼티가 반환한 노드는 텍스트 노드이거나 요소 노드다.
Element.prototype.firstElementChild	첫 번째 자식 요소 노드를 반환한다. firstElementChild 프로퍼티는 요소 노드만 반환한다.
Element.prototype.lastElementChild	마지막 자식 요소 노드를 반환한다. lastElementChild 프로퍼티는 요소 노드만 반환한다.

39.3.3 자식 노드 존재 확인

- 자식 노드가 존재하는지 확인하려면 Node.prototype.hasChildNodes 메서드를 사용한다.
- childNodes 프로퍼티와 마찬가지로 텍스트 노드를 포함하여 자식 노드의 존재를 확인한다.
- 자식 노드 중에 텍스트 노드가 아닌 요소 노드가 존재하는지 확인 하려면 children.length 또는 Element 인터페이스의 childElementCount 프로퍼티를 사용한다.

```
<!DOCTYPE html>
<html>
  <body>
    <ul id="fruits">
    </ul>
    <script>
      // 노드 탐색의 기점이 되는 #fruits 요소 노드를 취득한다.
      const $fruits = document.getElementById('fruits');

      // #fruits 요소에 자식 노드가 존재하는지 확인한다.
      // hasChildNodes 메서드는 텍스트 노드를 포함하여 자식 노드의 존재 확인한다.
      console.log($fruits.hasChildNodes()); // true

      // 자식 노드 중에 텍스트 노드가 아닌 요소 노드가 존재하는지 확인한다.
      console.log(!!$fruits.children.length); // 0 -> false
      console.log(!!$fruits.childElementCount); // 0 -> false
    </script>
```

```
  </body>
</html>
```

39.3.4 요소 노드의 텍스트 노드 탐색

- 요소 노드의 텍스트 노드는 자식 노드이므로 firstChild 프로퍼티로 접근할 수 있다.
- firstChild 프로퍼티가 반환한 노드는 텍스트 노드이거나 요소 노드이다.

```
<!DOCTYPE html>
<html>
  <body>
    <div id="foo">Hello</div>
    <script>
      // 요소 노드의 텍스트 노드는 firstChild 프로퍼티로 접근할 수 있다.
      console.log(document.getElementById('foo').firstChild); // #text
    </script>
  </body>
</html>
```

39.3.5 부모 노드 탐색

- 부모 노드를 탐색하려면 Node.prototype.parentNode 프로퍼티를 사용한다.
- 텍스트 노드는 DOM 트리의 최종단 노드인 리프 노드이므로 부모 노드가 텍스트 노드인 경우는 없다.

```
<!DOCTYPE html>
<html>
  <body>
    <ul id="fruits">
      <li class="apple">Apple</li>
      <li class="banana">Banana</li>
      <li class="orange">Orange</li>
    </ul>
    <script>
      // 노드 탐색의 기점이 되는 .banana 요소 노드를 취득한다.
      const $banana = document.querySelector('.banana');

      // .banana 요소 노드의 부모 노드를 탐색한다.
      console.log($banana.parentNode); // ul#fruits
    </script>
  </body>
</html>
```

39.3.6 형제 노드 탐색

- 부모 노드가 같은 형제 노드를 참색하려면 다음과 같은 노드 탐색 프로퍼티를 사용한다.
- 단, 어트리뷰트 노드는 요소 노드와 연결되어있지만 부모 노드가 같은 형제 노드가 아니기 때문에 반환되지 않는다.
- 즉, 아래 프로퍼티는 텍스트 노드 또는 요소 노드만 반환된다.

프로퍼티	설명
Node.prototype.previousSibling	부모 노드가 같은 형제 노드 중에서 자신의 이전 형제 노드를 탐색하여 반환한다. previousSibling 프로퍼티가 반환하는 형제 노드는 요소 노드뿐만 아니라 텍스트 노드일 수도 있다.
Node.prototype.nextSibling	부모 노드가 같은 형제 노드 중에서 자신의 다음 형제 노드를 탐색하여 반환한다. nextSibling 프로퍼티가 반환하는 형제 노드는 요소 노드뿐만 아니라 텍스트 노드일 수도 있다.

프로퍼티	설명
Element.prototype.previousElementSibling	부모 노드가 같은 형제 요소 노드 중에서 자신의 이전 형제 요소 노드를 탐색하여 반환한다. previousElementSibling 프로퍼티는 요소 노드만 반환한다.
Element.prototype.nextElementSibling	부모 노드가 같은 형제 요소 노드 중에서 자신의 다음 형제 요소 노드를 탐색하여 반환한다. nextElementSibling 프로퍼티는 요소 노드만 반환한다.

39.4 노드 정보 취득

- 노드 객체에 대한 정보를 취득하려면 다음과 같은 노드 정보 프로퍼티를 사용한다.

프로퍼티	설명
Node.prototype.nodeType	노드 객체의 종류, 즉 노드 타입을 나타내는 상수를 반환한다. 노드 타입 상수는 Node에 정의되어 있다. ▪ Node.ELEMENT_NODE: 요소 노드 타입을 나타내는 상수 1을 반환 ▪ Node.TEXT_NODE: 텍스트 노드 타입을 나타내는 상수 3을 반환 ▪ Node.DOCUMENT_NODE: 문서 노드 타입을 나타내는 상수 9를 반환
Node.prototype.nodeName	노드의 이름을 문자열로 반환한다. ▪ 요소 노드: 대문자 문자열로 태그 이름("UL", "LI" 등)을 반환 ▪ 텍스트 노드: 문자열 "#text"를 반환 ▪ 문서 노드: 문자열 "#document"를 반환

39.5 요소 노드의 텍스트 조작

39.5.1 nodeValue

- Node.prototype.nodeValue 프로퍼티는 setter와 getter 모두 존재하는 접근자 프로퍼티다. 따라서 참조와 할당 모두 가능.
- 노드 객체의 nodeValue 프로퍼티를 참조하면 노드 객체의 값을 반환한다.
- 노드 객체의 값이란 텍스트 노드의 텍스트다.
- 따라서 텍스트 노드가 아닌 노드, 즉 문서 노드나 요소 노드의 nodeValue 프로퍼티를 참조하면 null을 반환한다.

```
<!DOCTYPE html>
<html>
  <body>
    <div id="foo">Hello</div>
  </body>
    <script>
      // 문서 노드의 nodeValue 프로퍼티를 참조한다.
      console.log(document.nodeValue); // null

      // 요소 노드의 nodeValue 프로퍼티를 참조한다.
      const $foo = document.getElementById('foo');
      console.log($foo.nodeValue); // null

      // 텍스트 노드의 nodeValue 프로퍼티를 참조한다.
      const $textNode = $foo.firstChild;
      console.log($textNode.nodeValue); // Hello
    </script>
</html>
```

- 텍스트 노드의 nodeValue 프로퍼티에 값을 할당하면 텍스트 노드의 값, 즉 **텍스트를 변경할 수 있다**. 따라서 요소 노드의 텍스트를 변경하려면 다음과 같은 순서의 처리가 필요하다.
  - 텍스트를 변경할 요소 노드를 취득한 다음, 취득한 요소 노드의 텍스트 노드를 탐색한다.텍스트 노드는 요소 노드의 자식 노드이므로 firstChild 프로퍼티를 사용하여 탐색한다.
  - 탐색한 텍스트 노드의 nodeValue 프로퍼티를 사용하여 텍스트 노드의 값을 변경한다.

```
<!DOCTYPE html>
<html>
  <body>
    <div id="foo">Hello</div>
  </body>
    <script>
      // 1. #foo 요소 노드의 자식 노드인 텍스트 노드를 취득한다.
      const $textNode = document.getElementById('foo').firstChild;

      // 2. nodeValue 프로퍼티를 사용하여 텍스트 노드의 값을 변경한다.
      $textNode.nodeValue = 'World';
```

```
        console.log($textNode.nodeValue); // World
    </script>
</html>
```

39.5.2 textContent

- Node.prototype.textContent 프로퍼티는 setter와 getter 모두 존재하는 접근자 프로퍼티로서 요소 노드의 텍스트와 모든 자손 노드의 텍스트를 모두 취득하거나 변경한다.
- 요소 노드의 textContent 프로퍼티를 참조하면 요소 노드의 콘텐츠 영역 내의 텍스트를 모두 반환한다. 이때 HTML 마크업은 무시된다.

```
<!DOCTYPE html>
<html>
    <body>
        <div id="foo">Hello <span>world!</span></div>
    </body>
    <script>
        console.log(document.getElementById('foo').textContent); // Hello w
orld!
    </script>
</html>
```